CLASS NOTES

Models, Algorithms and Data: Introduction to computing

2019

Petros Koumoutsakos, Jens Honore Walther, Julija Zavadlav

(Last update: July 16, 2019)

IMPORTANT DISCLAIMERS

Much of the material (ideas, definitions, concepts, examples, etc) in these notes is taken for teaching purposes, from several references:

- Numerical Analysis by R. L. Burden and J. D. Faires
- The Nature of Mathematical Modeling by N. Gershenfeld
- A First Course in Numerical methods by U. M. Ascher and C. Greif

These notes are only informally distributed and intended ONLY as study aid for the final exam of ETHZ students that were registered for the course Models, Algorithms and Data (MAD): Introduction to computing 2019. The notes have been checked, however they may still contain errors so use with care.

LECTURE 3 Nonlinear systems II: set of equations

3.1 Introduction

We now want to generalize the discussion from the previous chapter to find the solution for a general system of N non-linear equations $f_i(\vec{x})$ $(i = 1,\dots,N)$, where $\vec{x} = (x_1,\dots,x_N)^T$ is a vector of N unknowns. We wish to find $\vec{x^*}$, such that

$$f_i(\vec{x^*}) = 0, \qquad i = 1,\dots,N \tag{3.1.1}$$

We write this system of equations as

$$\vec{F}(\vec{x}^*) = \begin{pmatrix} f_1(\vec{x}^*) \\ f_2(\vec{x}^*) \\ \vdots \\ f_N(\vec{x}^*) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \vec{0} \tag{3.1.2}$$

where $\vec{F} : \mathbb{R}^N \to \mathbb{R}^N$ and the components $f_i : \mathbb{R}^N \to \mathbb{R}$.

Example 3.1: Required pressure to sink an object

The amount of pressure required to sink a large heavy object in a soft homogeneous soil that lies above a hard base soil can be predicted by the amount of pressure required to sink smaller objects.

Let p denote the amount of pressure required to sink a circular plate of radius r at a distance d in the soft soil, where the hard soil lies at a distance $D > d$ below the surface. p can be approximated by an equation of the form:

$$p = k_1 e^{k_2 r} + k_3 r, \tag{3.1.3}$$

where k_1, k_2, k_3 depend on d and the consistency of the soil but not on r.

Now the task is to find k_1, k_2, k_3. To do that we need to have 3 equations which we can obtain by taking plates of different radii r_1, r_2, r_3 and sinking them. After doing so we will get the following

system of equations:

$$\begin{aligned} p_1 &= k_1 e^{k_2 r_1} + k_3 r_1, \\ p_2 &= k_1 e^{k_2 r_2} + k_3 r_2, \\ p_3 &= k_1 e^{k_2 r_3} + k_3 r_3. \end{aligned} \tag{3.1.4}$$

As in the previous Lecture, we will solve this system with iterative algorithms that given an initial guess $\vec{x^{(0)}}$ compute a sequence $\vec{x^{(1)}}, \vec{x^{(2)}}, \ldots, \vec{x^{(k)}}, \ldots$ that hopefully converge to the true solution $\vec{x^*}$. Note that if $N = 1$, our problem simplifies to the case of a single non-linear equation.

Some of the algorithms that we have encountered in the previous Lecture can be extended to a system of non-linear equations but not all. In particular, the Bisection method cannot be extended, but Newton's and Secant methods can be.

Taylor series for vector functions

To extend the algorithms for a system of non-linear equations we first need an extension of scalar Taylor Series theorem to vector functions.

If $\vec{x} = (x_1, \ldots, x_N)^T$ and $\vec{F} = (f_1, \ldots, f_N)^T$, which has bounded derivatives up to order at least two, then for a direction vector $\vec{y} = (y_1, \ldots, y_N)^T$ the Taylor expansion is given by

$$f_i(\vec{x} + \vec{y}) = f_i(\vec{x}) + \sum_{j=1}^{N} \frac{\partial f_i(\vec{x})}{\partial x_j} y_j + \mathcal{O}(\|\vec{y}\|^2) \tag{3.1.5}$$

or

$$\vec{F}(\vec{x} + \vec{y}) = \vec{F}(\vec{x}) + J(\vec{x})\vec{y} + \mathcal{O}(\|\vec{y}\|^2) \tag{3.1.6}$$

where J is the **Jacobian matrix**. It is an $N \times N$ matrix with elements $J(\vec{x})_{i,j} = \frac{\partial f_i(\vec{x})}{\partial x_j}$:

$$J(\vec{x}) = \begin{pmatrix} \frac{\partial f_1(\vec{x})}{\partial x_1} & \frac{\partial f_1(\vec{x})}{\partial x_2} & \cdots & \frac{\partial f_1(\vec{x})}{\partial x_N} \\ \frac{\partial f_2(\vec{x})}{\partial x_1} & \frac{\partial f_2(\vec{x})}{\partial x_2} & \cdots & \frac{\partial f_2(\vec{x})}{\partial x_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_N(\vec{x})}{\partial x_1} & \frac{\partial f_N(\vec{x})}{\partial x_2} & \cdots & \frac{\partial f_N(\vec{x})}{\partial x_N} \end{pmatrix} \tag{3.1.7}$$

Graphically, we can think of $\vec{x}$ as a point in $\mathbb{R}^N$ and of $\vec{y}$ as a direction vector. The point $\vec{x} + \vec{y}$ is obtained by moving from a point $\vec{x}$ in the direction $\vec{y}$. If we compare this Taylor series expansion with the one for scalar functions, i.e., $f(x + y) = f(x) + f'(x)y + \mathcal{O}(y^2)$ we see that the derivative f' is for vector functions replaced by the Jacobian matrix J.

Condition number

For a system of N non-linear equations and N unknowns, the condition number of the root finding problem for root $\vec{x}^*$ of $\vec{F}$ is $\|J^{-1}(\vec{x}^*)\|$ where J is the $N \times N$ Jacobian matrix.

3.2 Newton's method

As before, we derive the Newton's method by expanding $\vec{F}$ around $\vec{x}^*$

$$\overbrace{\vec{F}(\vec{x}^*)}^{0} \approx \vec{F}(\vec{x}^{(k)}) + J(\vec{x}^{(k)})(\vec{x}^* - \vec{x}^{(k)}), \tag{3.2.1}$$

where we have kept only two terms since we assume that $|\vec{x}^* - \vec{x}^{(k)}|$ is small. Solving for $\vec{x}^*$ gives us the expression for the update rule

$$\boxed{\vec{x}^{(k+1)} = \vec{x}^{(k)} - J^{-1}(\vec{x}^{(k)})\vec{F}(\vec{x}^{(k)})} \tag{3.2.2}$$

In practice, we never invert $J(\vec{x}^{(k)})$. Instead we solve

$$J(\vec{x}^{(k)})\vec{y}^{(k)} = -\vec{F}(\vec{x}^{(k)}) \tag{3.2.3}$$

for $\vec{y}^{(k)}$ and update as $\vec{x}^{(k+1)} = \vec{x}^{(k)} + \vec{y}^{(k)}$ as in the pseudocode Algorithm 1. To sum up: we start with an initial approximation $\vec{x}^{(0)}$ and iteratively improve the approximation by computing new approximations $\vec{x}^{(k)}$. The method succeeds when $\vec{x}^{(k)}$ is close to $\vec{x}^*$.
We note that for a linear system, we have $\vec{F}(\vec{x}) = A\vec{x} - \vec{b}$ and $J(\vec{x}) = A$. A single iteration of Newton's method then computes

$$\vec{x}^{(1)} = \vec{x}^{(0)} - J^{-1}(\vec{x}^{(0)})\vec{F}(\vec{x}^{(0)}) = \vec{x}^{(0)} - A^{-1}A\vec{x}^{(0)} + A^{-1}b = A^{-1}b, \tag{3.2.4}$$

which gives the same solution as when simply solving the system $A\vec{x} = \vec{b}$.

Convergence rate

The convergence of Newton's method for systems of equations is quadratic, provided that the Jacobian matrix is non-singular.

Computational Cost

For large N and dense Jacobians J, the cost of Newton's method can be substantial. It costs $O(N^2)$ to build the matrix J and solving the linear system in Eq. (3.2.3) costs $O(N^3)$ operations (if J is a dense matrix).

Algorithm 1 Newton's method

Input:

$\vec{x}^{(0)}$, {vector of length N with initial approximation}

tol, {tolerance: stop if $\|\vec{x}^{(k+1)} - \vec{x}^{(k)}\| < tol$}

k_{max}, {maximal number of iterations: stop if $k > k_{max}$}

Output:

$\vec{x}^{(k)}$, {solution of $\vec{F}(\vec{x}^{(k)}) = \vec{0}$ within tolerance tol} (or a message if $k > k_{max}$ reached)

Steps:

$k \leftarrow 0$

while $k \leq k_{max}$ **do**

  Calculate $\vec{F}(\vec{x}^{(k)})$ and $N \times N$ matrix $J(\vec{x}^{(k)})$

  Solve the $N \times N$ linear system $J(\vec{x}^{(k)})\vec{y} = -\vec{F}(\vec{x}^{(k)})$

  $\vec{x}^{(k+1)} \leftarrow \vec{x}^{(k)} + \vec{y}$

  if $\|\vec{y}\| < tol$ **then**

    break

  end if

  $k \leftarrow k + 1$

end while

Example 3.2: Geometric interpretation

We have N hyper-surfaces each with a tangent hyperplane (of dimension $N-1$) at $\vec{x}^{(k)}$. The surfaces and the 0 plane intersect at $\vec{x}^*$. The tangent planes, on the other hand, intersect with 0 plane at $\vec{x}^{(k+1)}$.

To understand this, consider the case of 2 equations ($f(x,y) = 0$ and $g(x,y) = 0$) and 2 unknowns (x,y). Then the equation $\vec{F}(\vec{x}^{(k)}) + J\left(\vec{x}^{(k)}\right)(\vec{x} - \vec{x}^{(k)}) = 0$ can be written as

$$\begin{pmatrix} f(x^{(k)}, y^{(k)}) \\ g(x^{(k)}, y^{(k)}) \end{pmatrix} + \begin{pmatrix} \frac{\partial f(x^{(k)}, y^{(k)})}{\partial x} & \frac{\partial f(x^{(k)}, y^{(k)})}{\partial y} \\ \frac{\partial g(x^{(k)}, y^{(k)})}{\partial x} & \frac{\partial g(x^{(k)}, y^{(k)})}{\partial y} \end{pmatrix} \begin{pmatrix} x - x^{(k)} \\ y - y^{(k)} \end{pmatrix} = 0$$

or using a short notation $f_x = \partial f(x^{(k)}, y^{(k)})/\partial x$, $f_y = \partial f(x^{(k)}, y^{(k)})/\partial y$

$$f_x(x - x^{(k)}) + f_y(y - y^{(k)}) + f(x^{(k)}, y^{(k)}) = 0$$
$$g_x(x - x^{(k)}) + g_y(y - y^{(k)}) + g(x^{(k)}, y^{(k)}) = 0$$

The two equations each represent a plane. The first equation defines a plane tangent to surface

$z = f(x,y)$ at point $\left(x^{(k)}, y^{(k)}, f\left(x^{(k)}, y^{(k)}\right)\right)$, while the second equation defines the tangent plane to $z = g(x,y)$. The intersection of this planes and $z = 0$ gives us the point $\left(x^{(k+1)}, y^{(k+1)}\right)$, i.e., the new approximation to the solution. If we compare this geometrical interpretation with the one for a single non-linear equation we see that a tangent line is here replaced by a tangent plane.

3.3 Other Methods

The cost of Newton's method can be reduced by:

- using function values at successive iterations to build approximate Jacobians and avoid explicit evaluations of the derivatives (note that this is also necessary if for some reason you cannot evaluate the derivatives analytically)
- update factorization (to solve the linear system) of approximate Jacobians rather than refactoring it each iteration

Modified Newton Method

The simplest approach is to keep J fixed during the iterations of Newton's method. So instead of updating or recomputing Jacobian matrix we compute it once $J^{(0)} = J(x^{(0)})$ and (instead of Eq. (3.2.3)) solve

$$J^{(0)}\vec{y}^{(k)} = -\vec{F}(\vec{x}^{(k)}). \tag{3.3.1}$$

This can be done efficiently by performing a LU decomposition of $J^{(0)}$ once and use it over and over again for different $\vec{F}(\vec{x}^{(k)})$. This removes the $O(N^3)$ cost of solving the linear system. We only have to evaluate $\vec{F}(\vec{x}^{(k)})$ and can then compute $\vec{y}^{(k)}$ in $O(N^2)$ operations. This method can only succeed if J is not changing rapidly.

Quasi Newton Method

A method in between Newton and modified Newton is the quasi Newton method which *changes* J at every step without recomputing the derivatives $\partial f_i(\vec{x})/\partial x_j$ (see Eq. (3.1.7)). It instead updates $J^{(0)} = J(x^{(0)})$ by using information from the step itself.

After the first step we know: $\Delta\vec{x} = \vec{x}^{(1)} - \vec{x}^{(0)}$ and $\Delta\vec{F} = \vec{F}(\vec{x}^{(1)}) - \vec{F}(\vec{x}^{(0)})$. So derivatives of the f_i are in the direction of $\Delta\vec{x}$. Then the next $J^{(1)}$ is adjusted to satisfy:

$$J^{(1)}\Delta\vec{x} = \Delta\vec{F} \tag{3.3.2}$$

for example by the rank-1 update

$$J^{(1)} = J^{(0)} + \frac{\left(\Delta\vec{F} - J^{(0)}\Delta\vec{x}\right)\left(\Delta\vec{x}\right)^T}{\left(\Delta\vec{x}\right)^T\left(\Delta\vec{x}\right)}. \tag{3.3.3}$$

The advantage of the rank-1 update is that it allows the LU decomposition of $J^{(1)}$ to be computed in $O(N^2)$ given the LU decomposition of $J^{(0)}$. As in the modified Newton's method, this essentially reduces the cost of solving the linear system from $O(N^3)$ to $O(N^2)$.

3.4 Non-Linear Optimization

Can we use the root-finding algorithms to solve an optimization problem? Let's consider a minimization problem

$$\min_{\vec{x}} E(\vec{x}), \tag{3.4.1}$$

where we have N unknowns $\vec{x} = (x_1, \ldots, x_N)^T$ and function $E : \mathbb{R}^N \to \mathbb{R}$ is a scalar function of N variables. Note, that a maximization problem $\max_{\vec{x}} E(\vec{x})$ is equivalent to a minimization $\min_{\vec{x}}[-E(\vec{x})]$.

Sufficient condition for a critical point of $E(\vec{x})$ at $\vec{x}^*$ is

$$\nabla E(\vec{x}^*) = \begin{pmatrix} \frac{\partial E(\vec{x}^*)}{\partial x_1} \\ \frac{\partial E(\vec{x}^*)}{\partial x_2} \\ \vdots \\ \frac{\partial E(\vec{x}^*)}{\partial x_N} \end{pmatrix} = \vec{0}, \tag{3.4.2}$$

where $\nabla E(\vec{x})$ is a gradient vector of first derivatives of E at $\vec{x}$. The critical point is a minimum if

$$\nabla^2 E(\vec{x}^*) = H = \begin{pmatrix} \frac{\partial^2 E(\vec{x}^*)}{\partial x_1^2} & \frac{\partial^2 E(\vec{x}^*)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 E(\vec{x}^*)}{\partial x_1 \partial x_N} \\ \frac{\partial^2 E(\vec{x}^*)}{\partial x_2 \partial x_1} & \frac{\partial^2 E(\vec{x}^*)}{\partial x_2^2} & \cdots & \frac{\partial^2 E(\vec{x}^*)}{\partial x_2 \partial x_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E(\vec{x}^*)}{\partial x_N \partial x_1} & \frac{\partial^2 E(\vec{x}^*)}{\partial x_N \partial x_2} & \cdots & \frac{\partial^2 E(\vec{x}^*)}{\partial x_N^2} \end{pmatrix} \tag{3.4.3}$$

is a positive definite matrix. $\nabla^2 E(\vec{x})$ is called a **Hessian matrix** H, which contains second derivatives of E at $\vec{x}$. Note, that for a scalar function $E(x)$ with 1 variable x the condition for a minimum at x^* reduces to $E'(x^*) = 0$ and $E''(x^*) > 0$.

The condition for a critical point yields a system of nonlinear equations

$$\vec{F}(\vec{x}) = \nabla E(\vec{x}) = \vec{0}, \tag{3.4.4}$$

that we can solve with root-fining algorithms discussed before, e.g., with Newton's method.

Newton's method

The Jacobian matrix $J(\vec{x})$ is equal to the Hessian matrix of E

$$J(\vec{x}) = \nabla^2 E(\vec{x}). \tag{3.4.5}$$

The Newton's update rule is therefore,

$$\begin{aligned} \nabla^2 E(\vec{x}^{(k)})\vec{y}^{(k)} &= -\nabla E(\vec{x}^{(k)}) \\ \vec{x}^{(k+1)} &= \vec{x}^{(k)} + \vec{y}^{(k)} \end{aligned} \tag{3.4.6}$$

Steepest descent / gradient descent method

Newton's method has several difficulties: it requires evaluation of the Hessian matrix and a solution of linear system of equations at each iteration, far form the minimum it is unreliable and can diverge from minimum. A simpler method is to move in the local gradient descent direction:

$$\begin{aligned} \vec{y}^{(k)} &= -\nabla E(\vec{x}^{(k)}) \\ \vec{x}^{(k+1)} &= \vec{x}^{(k)} + \eta\vec{y}^{(k)} \end{aligned}$$

With this method, we do not need to evaluate the Hessian matrix nor do we need to solve a linear system of equations at each iteration step. However, we must determine the appropriate step size η and the convergence is not as fast as with Newton. Gradient based techniques will always find minima that are close to the starting point (these might be local and not a global minimum).

Levenberg-Marquardt Method

A combination of both methods is proposed by Levenberg-Marquardt method, where far away from minimum a Gradient Descent is performed while close to the minimum a Newton's method is used. With such an interpolates one can overcome the possible divergence of Newton's method. Here we have,

$$\begin{aligned} (\nabla^2 E(\vec{x}^{(k)}) + \lambda I)\vec{y}^{(k)} &= -\nabla E(\vec{x}^{(k)}) \\ \vec{x}^{(k+1)} &= \vec{x}^{(k)} + \vec{y}^{(k)}, \end{aligned}$$

where λ is a damping parameter that is adjusted at each iteration step. Small λ the algorithm is essentially Newton's method while for large λ the update is in the gradient descent direction. We start with large λ. Then, if the step decreases $E(\vec{x}^{(k)})$, λ is decreased. If the step increases the error, λ is increased. The Levenberg-Marquardt method is primarily used for the non-linear Least Squares problem.

3.4.1 Non-Linear Least Squares

In the first lecture, we considered a linear Least Squares problem of function fitting where given a set of data $\{t_i, b_i\}_{i=1}^N$ we computed the M unknown parameters in $\vec{x}$ by solving the system $A\vec{x} \approx \vec{b}$. The

fitting function $f(t)$ had parameters $\{x_m\}_{m=1}^M$ that appeared outside of the basis functions. Now, we consider a case where the unknown parameters appear inside the basis functions, i.e.,

$$f(t_i) = \sum_{k=1}^{K} \phi_k(t_i; \vec{x}_k). \tag{3.4.7}$$

Note that now we may have more parameters $\{x_m\}_{m=1}^M$ than basis functions. The cost function can still be expressed as:

$$E^2 = \sum_{i=1}^{N} [b_i - f(t_i; \vec{x})]^2 = \left\| \vec{b} - \vec{F}(t; \vec{x}) \right\|^2 \tag{3.4.8}$$

We wish to find the $\bar{x}$ which minimizes $E(\vec{x}) : \mathbb{R}^M \to \mathbb{R}$.

Example 3.3: Gaussian basis functions

Consider fitting a Gaussian function to data where the unknown parameters are the amplitude α, location μ and variance σ.

$$\phi(t; \alpha, \mu, \sigma) = \alpha \exp\left(\frac{(t-\mu)^2}{2\sigma^2} \right)$$

Here $\vec{x}$ includes all the coefficients to determine for ϕ, i.e., $\vec{x} = (\alpha_k, \mu_k, \sigma_k)^T$. Note that fixing μ, σ would result in a linear model.

To employ **Newton's method** we need to evaluate the terms ∇E^2 and $\nabla^2 E^2$ that appear in Eq. (3.4.6). The first term is equal to

$$\begin{aligned} \frac{\partial E^2}{\partial x_k} &= -2 \sum_{i=1}^{N} [b_i - f(t_i; \vec{x})] \frac{\partial f(t_i; \vec{x})}{\partial x_k} \\ \nabla E^2 &= -2 J(\vec{x})^T [\vec{b} - \vec{F}(t; \vec{x})] \end{aligned} \tag{3.4.9}$$

where $J(\vec{x})$ is a NxM Jacobian matrix of $\vec{F}$. The second term is

$$\begin{aligned} \frac{\partial^2 E^2}{\partial x_k \partial x_l} &= -2 \sum_{i=1}^{N} \left\{ -\frac{\partial f(t_i; \vec{x})}{\partial x_k} \frac{\partial f(t_i; \vec{x})}{\partial x_l} + [b_i - f(t_i; \vec{x})] \frac{\partial^2 f(t_i; \vec{x})}{\partial x_l \partial x_k} \right\} \\ \nabla^2 E^2 &= -2[-J(\vec{x})^T J(\vec{x}) + L(\vec{x})] \end{aligned} \tag{3.4.10}$$

where $L(\vec{x}) = \sum_{i=1}^{N} [b_i - f(t_i; \vec{x})] \frac{\partial^2 f(t_i; \vec{x})}{\partial x_l \partial x_k}$. Plugging this expressions into Eq. (3.4.6) gives us

$$\begin{aligned} \left[J(\vec{x}^{(k)})^T J(\vec{x}^{(k)}) - L(\vec{x}^{(k)}) \right] \vec{y}^{(k)} &= J(\vec{x}^{(k)})^T [\vec{b} - \vec{F}(t; \vec{x}^{(k)})] \\ \vec{x}^{(k+1)} &= \vec{x}^{(k)} + \vec{y}^{(k)} \end{aligned} \tag{3.4.11}$$

Since $L(\vec{x}^{(k)})$ is proportional to the residual between the model and the the data it should eventually be small and can therefore be omitted. This would give us the **Gauss–Newton method**.

For the **Steepest descent method**, we would have

$$\begin{aligned} \vec{y}^{(k)} &= J(\vec{x}^{(k)})^T[\vec{b} - \vec{F}(t;\vec{x}^{(k)})] \\ \vec{x}^{(k+1)} &= \vec{x}^{(k)} + \eta\vec{y}^{(k)} \end{aligned} \tag{3.4.12}$$

Exam checklist

- What are the key algorithms for solving a system of non-linear equations
- What is the Jacobian for Newton's method and what is its role
- Why do we approximate the Jacobian in Newton's methods
- How many solutions is a nonlinear system of n algebraic equations expected to have?
- Explain how we can use the algorithms for solving a system of non-linear equations in optimization problem
- What are the necessary and the sufficient conditions for a minimum?
- Can we use Newton's method to solve Least Squares problem?